<알고리즘 Mix Tape 04>

9장 문자열 매칭 알고리즘(170614)

이제 어느덧 기말고사이다. 여러분들이 알고리즘을 공부하는데 있어서 초반에는 어려움이 많았지만, 나중에는 교수님이 강의하지 않은 부분에 대해서도, 요약정리에서도 언급하지 않은 개념들에 대해서도 공부를 따로 해볼 수 있는 기회를 가져봤으면 좋겠다. 우리가 과거에 C언어에서 문자(char 형) 배열에 대해 생각을 해봤을 것이다. 여기에 들어있는 알파벳(혹은 숫자, 특수기호)을 비교해서 문자열에 사용자가 원하는 단어가 포함이 되어있으면 그 인덱스가 들어있다는 증거를 댈 수 있도록 보여주는 것이 바로 문자열 매칭 알고리즘의 실질적인 목표이다. 실제로 삼성 코드그라운드 알고리즘 문제에서도 이 개념이 나온 적도 있으니 이 개념에 대해서 자세히 공부해보는 기회를 가져보도록 하자.

1. 문자열 매칭의 기초

필자는 음악을 좋아한다. 특히 랩이랑 옛날 노래를 듣는 경우가 대다수인데 랩을 사례로 본다면 반복이 되는 구절이 있을 것이다. 예를 들어 랩이나 노래 하나를 집어서 접근을 해보도록 하자.

> **이 생에 못한** 사랑 **이 생에 못한** 인연
> 먼길 돌아 다시 만나는 날 나를 놓지 말아요
> - 이선희, 인연

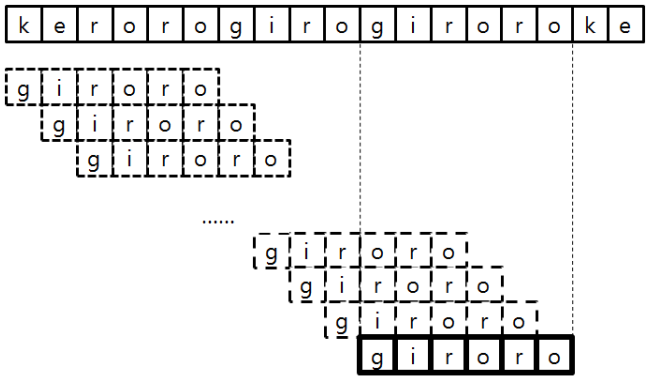
> 난 **연예인**도 아닌데 **연예인**보다 유명한 Rap Star
> 모른다면 친구한테 **물**어봐
> 그 담엔 내가 냈던 노랠 **들**어봐
> Illionaire Baby 그렇게 **물들**어가
> - Dok2, 111%

여기 가사에서 같은 키워드에 대해서 나오는 경우가 있는데 이 생에 못한 이나 연예인이 2번 나오는 반복을 이용을 통해서 랩에서 말하고자 하는 내용을 강조를 하거나 아래처럼 물, 들을 통해서 마지막 마디에 물들로 통해서 숨겨진 라임을 보여주는 경우가 있다. 실제로 문자열 내부에서도 사용자가 특정한 단어를 반복하는 경우를 통해서 위치를 출력하도록 하는데 이는 실제로도 Ctrl+F 키 눌러서 키워드를 찾는 원리를 고려해볼 수 있다.

이러한 원리에 대해서 문자열 내부에서 원하는 단어를 찾는 알고리즘을 연습해보도록 하자.

```
static int findIndex(String A, String P) {
  int n = A.length(), m = P.length();
  for (int i = 0; i <= n - m; ++i) { // 1
    boolean 일치 = true;
    for (int j = 0; j < m; ++j) // 2
      if (A.charAt(i + j) != P.charAt(j)) {
        일치 = false;
        break;
      }
    if (일치) return i; // 일치하는 부분 문자열을 찾은 경우에 시작 위치(index)를 리턴
  }
  return -1; // 일치하는 문자열을 찾지 못한 경우에 -1을 리턴
}
public static void main(String[] args) {
  String s = "kerorogirogiroroke";
  System.out.printf("%d %s\n", findIndex(s, "keroro"), s.indexOf("keroro"));
  System.out.printf("%d %s\n", findIndex(s, "giroro"), s.indexOf("giroro"));
  System.out.printf("%d %s\n", findIndex(s, "dororo"), s.indexOf("dororo"));
  System.out.printf("%d %s\n", findIndex(s, "kuroro"), s.indexOf("kuroro"));
}
```

문자열의 매칭을 쉽게 설명하기 위하여 잠깐 케로로를 인용하도록 하자. 일단 s라는 엄청 긴 문자열에 대해서 giroro는 어떻게 찾을 것인지 감이 안 잡힐 것이다. 일단 초딩적인 방법으로 접근을 해보도록 하자.



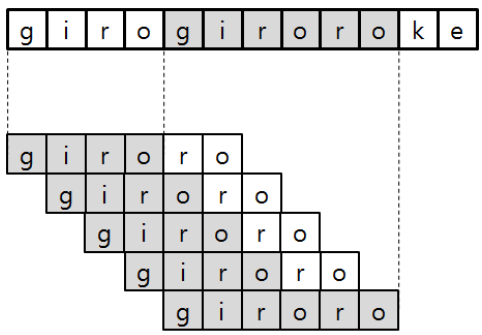
일단 s의 글자 수는 18개이다. 그리고 giroro의 글자 수는 총 6개이다. 이처럼 문자열을 매칭해서 찾기 위하여 13번을 갖다 대고 비교를 해야 하니 시간 복잡도가 장난 아니게 복잡하게 느껴질 것이다. 쉽게 생각해보면 큰 문자열 중 인덱스 별로 글자 6개를 잡아서 검색을 원하는 문자열에 대해 비교를 하게 된다. 이 문장에 대한 자세한 설명은 크게 필요 없이 쉽게 이해할 것이다. 그러면 이 문장의 시간 복잡도는 어떻게 될까?

-> 2번 문장 이내에서 이루어지는 시간 복잡도는 O(m)이 된다.

-> 1번 문장 이내에서 반복되는 횟수는 n-m+1이 되기 때문에 실상 시간 복잡도는 O(n-m)이 나오게 된다. 그럼 2번 문장과 1번 문장을 콜라보 때려보자.

-> $nm-m^2$ 이 나오게 되겠지만, 여기서 n이란 변수는 긴 문자열이기 때문에 당연한 이야기로 n>m이기 때문에 최종적으로 시간 복잡도는 $O(mn)$이 나오게 된다.

그런데 이렇게 문자열을 비교하는 것을 본다면 뭔가 바보같다는 생각이 들 것이다. 예를 들어 아래와 같은 경우를 살펴보자.



무엇을 느끼는가? 바로 giroro 단어를 비교하는 것 뿐인데 girogi... 이런 식으로 있어서 계속 비교를 하기엔 뻘짓으로 느낄 수 밖에 없다. 그래서 이를 보완하기 위해서 오토마타에 대해 공부해보도록 하자.

2. 오토마타(Automata)

오토마타라는 것은 원래 문자열 매칭을 위한 알고리즘으로 받아들일 수 있겠는데 실제로는 문제 해결 절차를 상태의 전이로 나타내는 개념으로 문자열 비교 이외에 암호 해석 등등에도 쓰이는 개념으로 생각할 수 있다. 오토마타의 구성 요소에는 어떠한 것들이 존재할까?

$(Q, q_0, A, \Sigma, \delta)$

-> Q는 상태의 집합으로서 해석 결과를 판단하는데 이용된다.

-> q_0는 시작 상태로 생각을 하면 되겠다.

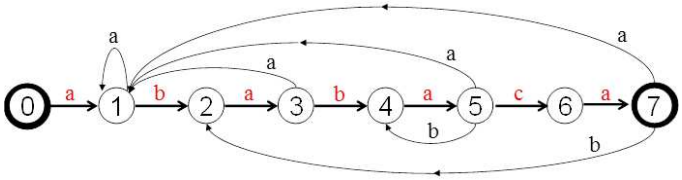
-> A는 목표 상태들의 집합으로 생각을 하면 되는데 목표를 도달하기 위하여 여러 번 도달을 하게 되면 집합의 모양이 달라진다.

-> Σ는 입력 알파벳의 집합이 되겠는데 표준 영어의 알파벳은 26자(대문자까지 하면 52자)가 되지만 한글(가나다), 일본어(ようこそ)까진 괜찮은데 중국어(你好吗?)나 한자(실제로 실용 한자만 해도 3500자인데 더 더욱 많을 것이다.)로 해버리면 오토마타가 복잡해진다.

실제로 오토마타를 이용하게 되면 어느 부분에서 건너뛰는지에 대해 자동적으로 계산을 때려주는데 이 오토마타에 대해 어떤 원리로 접목을 해야 할지에 대한 생각을 짚고 넘어가보도록 하자.

2-1) 단어를 오토마타 배열로 구현하기

지금 알고리즘 시점에서 우리말로 접근을 하기에 방대한 어려움이 따르기 때문에 ASCII 코드 범위 이내에 있는 알파벳들 중의 일부로 사례를 들어보자. 쉽게 생각을 해보기 위해서 알파벳은 a에서 c까지 범위로 잡아서 생각을 해보도록 하자. 문자열에서 찾기 원하는 단어가 abababaca인 경우에는 오토마타를 어떻게 형성을 해야 할까? 실제로 오토마타를 만드는 알고리즘이 따로 존재하는데 이에 대해서는 강의노트에 있는 오토마타를 통해서 어떠한 알고리즘으로 돌아가는지에 대해 설명을 하겠다.



일단 컴퓨터의 지식으로서는 사람들이 쓰는 말에 대하여 분별을 못한다. 그래서 사람처럼 어느 부분이 반복이 되는지에 대해서 인지를 할 수 없다. 이를 해결하기 위해 만든 개념이 바로 오토마타로 생각을 하면 되는데 상태를 전이하면서 어떤 원리로 돌아가는지에 대해서만 익혀보도록 하자. 실제로 오토마타를 만드는 알고리즘을 살펴보면 꽤 어렵기 때문에 문자열 내부에서 단어를 찾을 때 효율적으로 쓰이는구나 라는 의미를 새기고 넘어가자.

0->1로 전이 : 문자열 중에 인덱스 별로 찾게 되면 a가 나오면 상태를 1로 전이를 하도록 한다.

1번 상태에서 : b가 나오면 2번 상태로 넘어가는데 여기서 a가 나오면 1번 상태로 돌아가면 된다.

2번 상태에서 : a를 만나면 3번으로 돌아가게 되는데 전이되는 선 이외의 알파벳을 만나면 당연히 0번으로 돌아가는 것은 알고 넘어가도록 하자.

3번 상태에서 : 문자 a를 만나면 1번 상태로 넘어가게 되는데 문자 b를 만나면 4번 상태로 넘어가게 되고, 나머지 문자열에 대해 만나면 0번으로 넘어가게 된다.

실제로 오토마타를 이용해서 비교를 한다면 상태란 변수를 이용해서 쉽게 비교를 할 수 있다는데 어떤 면에서 비교를 할 수 있다는 뜻일까? 아래 알고리즘을 보면서 알아보도록 하자.

```
public class Example2a {
  static int[] alphabet = {0, 1, 2, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3};
  static int[][] automata = {
    {1, 0, 0, 0},
    {1, 2, 0, 0},
    {3, 0, 0, 0},
    {1, 4, 0, 0},
    {5, 0, 0, 0},
    {1, 4, 6, 0},
    {7, 0, 6, 0},
    {1, 2, 0, 0}
  };
  static int findIndex(String A, String P) {
    int state = 0, n = A.length(), m = P.length();
    for (int i = 0; i < n; ++i) {
      int c = A.charAt(i);
      int index = alphabet[c - 'a'];
      state = automata[state][index]; // 1
      if (state == m) return i - m + 1;
    }
    return -1; // 일치하는 문자열을 찾지 못한
경우에 -1을 리턴
  }
  public static void main(String[] args) {
   String A = "abcdefabdafaababacabbb";
   String P = "abababaca";
   System.out.println(findIndex(A, P));
   System.out.println(A.indexOf(P));
  }
}
```

이 문장들을 이용해서 A이란 방대한 문자열 내부에서 단어 P를 찾는 알고리즘을 작동하게 된다면 우선 아래와 같이 돌아갈 것이다.

case01) a~f까지 6글자에서 살펴본 state

	a	b	c	d	e	f
state	1	2	0	0	0	0

일단 위에서 만들어진 오토마타를 기준으로 살펴본다면 a로 시작을 하니 오토마타의 state는 1이 된다. 그리고 b까지 올라오게 되면 state는 2로 급상하게 된다. 하지만 다음에 state가 3으로 급상을 해야 하기 위해 a가 나와야 되는데 c가 나왔다. 그럼 우짤 수 없이 state는 0으로 돌아가게 된다. 그 후로는 a이란 문자가 나올 때까지 상태는 0으로 유지된다.

case02) abdafa까지 살펴본 state

	a	b	d	a	f	a
state	1	2	0	1	0	1

여기서도 마찬가지로 a, b까지는 정상적으로 오토마타의 상태가 올라가게 되는데 d라는 녀석을 만나게 되어 애석하게도 0으로 떨어지고 말게 된다. 그 다음 a, f, a가 나오게 되면 상태는 아직까지 1을 유지하게 되고 만다.

case03) babaca까지 살펴본 state

	b	a	b	a	c	a
state	2	3	4	5	6	7

이제 a 다음에 b를 만나면 2로 급상하고, b 다음 a를 만나면 3으로 급상하고, 방금 설명한 오토마타를 통해서 계속 급상을 하는 상태로 작동을 하게 되니깐 결국에 state가 7이 나오게 되고, 최종적으로 단어로 만들어진 오토마타에 대해 만족을 하게 되면서 반복문을 벗어나게 된다. 여기서 state랑 단어의 글자가 같다는 뜻이 곧 단어의 인덱스를 찾았다는 의미로 받아들여야 하며, (현재 인덱스)-(단어 글자 수)+1로 결과를 반환하면 되겠다.

그런데 하필이면 음영 친 부분이 이상하게 생각될 것이다. 왜 중간에 알파벳 배열은 무엇이며, 오토마타의 배열은 열의 수가 짧아졌을까? 오토마타에서 만들어지는 단어 내부에서 쓰이는 언어에 대해 고려를 할 수 있는데 예를 들어 여기서는 소문자 알파벳으로만 구성이 되었는데 a, b, c 이렇게 3가지만 쓰였다. 그래서 알파벳 별로 a는 0번 인덱스, b는 1번 인덱스, c는 2번 인덱스, d부터 다른 알파벳들은 3번 인덱스를 통하여 오토마타의 규모를 줄여서 공간 복잡도를 아끼려는 목적으로 작성을 하였다.

2-2) 오토마타 문자열 매칭 시간 복잡도는?
실제로 이 문장을 살펴보게 되면 시간 복잡도가 O(n)이구나 라고 느끼는 사람들도 있는데 여기서는 다루지 않았는데 우리가 요리를 하기 앞서서 재료들에 대해서 전처리를 해두는 경우를 살펴본 적이 있다. 그래서 찾을 단어에 대한 오토마타를 전처리를 하는 것도 고려를 해두면 원래 시간 복잡도는 O(m*|∑|)이 나오게 된다. 여기서 m은 단어의 수가 되겠고, 시그마는 단어에 쓰이는 언어의 문자들의 수로 살펴보면 되겠다. 그럼 이 두 시간 복잡도를 합치게 되면 O(m*|∑|)+O(n)이 나오게 된다. 여기서 m은 n보다 항상 작기 때문에 O(n)으로 해도 되나요 라고 생각하는 사람도 있는데 오토마타 전처리 작업을 통해서 문자열 매칭을 하기 때문에 이렇게 포함을 시켜둬야겠다.

3. KMP 알고리즘

여기서 KMP는 무슨 뜻일까? Knuth-Morris-Pratt 알고리즘의 줄임말인데 무슨 사람 이름 같기도 하고 이상한 이름인데 영화사 MGM(Metro Goldwyn Mayer)처럼 만들어진 이름으로 생각하고 넘어가자. 이 KMP 알고리즘은 오토마타를 이용한 매칭하는 과정과 유사한 알고리즘인데 여기서 중요한 것은 동일한 문자열이 나오게 된다면 어느 부분에서 스킵을 해야 할지에 대해 미리 계산을 해주는 것을 계산해두기 때문에 반복되는 단어에 대해서 굳이 state를 얻을 필요까지 없게 된다. 예를 들어 bcdbcdab란 문자열이 있다고 가정하자. 그러면 아래와 같은 배열이 완성이 되겠다.

-1	0	0	0	1	2	3	0	1

이 배열이 뜻하는 의미로서는 우리가 초등학교 때 간단한 동요로 돌림 노래를 해본 적이 있을 것이다. 같은 알파벳이 나오면 이전에 있던 알파벳을 가리키게 되면서 탐색을 하는데 있어서 쓸데없는 반복은 피하자는 뜻에서 만들어졌다고 생각해볼 수 있다. 그럼 이 알고리즘의 원리는 어떻게 되어있을까? 아래를 살펴보도록 하자.

```
public class Example3 {

    static int[] preprocessing(String P) {
        int[] pi = new int[P.length() + 1];
        int j = 0, k = -1;
        pi[0] = -1;
        while (j < P.length()) {
            if (k == -1 || P.charAt(j) ==
P.charAt(k)) {
                ++j; ++k;
                pi[j] = k;
            } else
                k = pi[k];
        }
        return pi;
    }

    static int findIndex(String A, String P) {
        int n = A.length(), m = P.length();
        int[] pi = preprocessing(P);
        int i = 0, j = 0;
        while (i < n) {
            if (j == -1 || A.charAt(i) ==
P.charAt(j)) {
                ++i; ++j;
            } else
                j = pi[j];
            if (j == m)
                return i - m; // 매칭 발견
        }
        return -1; // 일치하는 문자열을 찾지 못한
경우에 -1을 리턴
    }

    public static void main(String[] args) {
        String A = "abcdbcdadbcdbcdab";
        String P = "bcdbcdab";
        System.out.println(findIndex(A, P));
        System.out.println(A.indexOf(P));
    }
}
```

일단 여기서 n은 문자열의 글자 수로 알 수 있고, m은 단어의 글자 수로 알 수 있다. 그리고 j는 방금 오토마타를 통해서 그 상태로 돌아갈 수 있게끔 계산을 한 것으로 살펴보면 되겠다. 돌아가는 원리는 방금 오토마타랑 같은 원리로 돌아가게 되는데 여기서는 반복의 수만 줄어들게끔 하는 원리로 작동을 하였기 때문에 반복되는 단어에 대한 탐색을 줄이는 시점으로 살펴보면 되겠다. 그럼 이 문장의 시간 복잡도는 O(m)+O(n)이 되겠다. 이것도 전처리를 포함시켜야 하기 때문이다.